Copy elision revisited[1]

Copy elision is a common compiler optimization which allows the compiler to avoid unnecessary temporary objects even if copy construction has side effects. See [wiki:rvo], [blog-knatten] and [rvo-cargo-cult] for a more in-depth explanation.
Lots of articles concerning copy elision, also termed return value optimization (RVO), tell you that it is really fast to return huge objects from functions. The purported reason is that the compiler will usually elide the copy away. The important term here is *usually* as there are a lot of real world situations where the copy will not be elided away. In this short article I want exercise some of the failing cases.

Test setup

The code used for this project can be found at github [repository]. In the code a large object is returned from different functions, and the method how the object is returned is logged.
The entries in the table on page 3 are one of:

- `e` Copy elided by the compiler. Best case scenario as there is no runtime overhead.
- `m` Move construction. This is the second best option as it is equal or less work than calling the copy constructor. Only available for C++-11.
- `c` The copy constructor was called. The worst case from a performance point of view.

Remarks

Unnamed return value optimization (URVO) The easiest form of copy elision where the function returns an unnamed object. The first six functions `A-F` exercise that and all compilers succeed in eliding the copy.

Named return value optimization (NRVO) The functions `G-K` have a single return statement returning a named object. Only two compilers, Sun C++ and Embarcadero C++ (x86), fail completely in this part, with everyone else being at or close to optimum elision. The function `H` was introduced after the author found [gcc-bug:51571]. This case is especially interesting as icc has the same missed optimization bug as gcc.
Multiple return paths are created by either having a constant local boolean variable or by accepting a boolean parameter. In both cases the code is designed in a way that a perfect compiler could reduce the functions to have only one return path.
For the functions with zero parameters, functions `L`, `M`, `P`, `Q`, the situation is mixed. Gcc, icc and Digital Mars elide the copies in `L` and `P`, where all others create copies. On the other hand for `M` the best possible solution seems to be moving the result, whereas `Q` only provokes a copy for Sun C++ and Embarcadero (x86).

[1]

The functions accepting a boolean parameter, where in all cases a literal true was passed, can be split into the groups `R-S` and `N-O`. The first group will return in both return paths the identical named object. Here the clear winner is gcc with all copies being elided, most of the other compilers succeed only for `S` with the ternary operator being again the inhibitor of copy elision. In the second group `N-O` the situation is completey different. As the compiler now would have to reason across functions, none of the compilers is able to elide the copy. Worse `N` is the only function where every compiler generated a copy. For `O` the C++-11 compilers could at least move the resulting object.

Conclusions

From these tests a few observations can be made:

- Always return unnamed objects as all tested compilers will elide the copies (URVO).
- Don't try to outsmart the compiler with guidelines a la "never use the ternary operator/if statement for compiler foo'. The guidelines will be too difficult to remember and might change with the next compiler version.
- Compilers don't exercise NRVO hard enough which hinders more natural C++-APIs.

Feedback

For all kinds of comments/remarks/bug reports you can reach me at `thomas.braun@byte-physics.de`.

	nrvo_single()	nrvo_two()	nrvo_two_with_param(true)	nrvo_with_exception_1(true)	nrvo_with_exception_2(true)	nrvo_with_exception_3()	nrvo_single_1()	nrvo_single_2()	nrvo_single_with_exception_1(true)	nrvo_single_with_exception_2(true)	nrvo_single_with_exception_3()	nrvo_two_different_tern()	nrvo_two_different_if()	nrvo_two_different_with_param_tern(true)	nrvo_two_different_with_param_if(true)	nrvo_two_equal_tern()	nrvo_two_equal_if()	nrvo_two_equal_with_param_tern(true)	nrvo_two_equal_with_param_if(true)
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
gcc 3.4.6	e	e	e	e	e	e	e	c	e	e	e	e	c	c	c	e	e	e	e
gcc 4.0.4	e	e	e	e	e	e	e	c	e	e	e	e	c	c	c	e	e	e	e
gcc 4.1.2	e	e	e	e	e	e	e	c	e	e	e	e	c	c	c	e	e	e	e
gcc 4.2.4	e	e	e	e	e	e	e	c	e	e	e	e	c	c	c	e	e	e	e
gcc 4.3.6	e	e	e	e	e	e	e	m	e	e	e	e	m	c	m	e	e	e	e
gcc 4.4.7	e	e	e	e	e	e	e	m	e	e	e	e	m	c	m	e	e	e	e
gcc 4.5.4	e	e	e	e	e	e	e	m	e	e	e	e	m	c	m	e	e	e	e
gcc 4.6.4	e	e	e	e	e	e	e	m	e	e	e	e	m	c	m	e	e	e	e
gcc 4.7.3	e	e	e	e	e	e	e	m	e	e	e	e	m	c	m	e	e	e	e
gcc 4.8.5	e	e	e	e	e	e	e	m	e	e	e	e	m	c	m	e	e	e	e
gcc 4.9.3	e	e	e	e	e	e	e	m	e	e	e	e	m	c	m	e	e	e	e
gcc 5.1.0	e	e	e	e	e	e	e	m	e	e	e	e	m	c	m	e	e	e	e
gcc 5.2.0	e	e	e	e	e	e	e	m	e	e	e	e	m	c	m	e	e	e	e
gcc trunk 20150802	e	e	e	e	e	e	e	m	e	e	e	e	m	c	m	e	e	e	e
clang 3.1	e	e	e	e	e	e	e	e	e	e	e	c	m	c	m	c	e	c	e
clang 3.2	e	e	e	e	e	e	e	e	e	e	e	c	m	c	m	c	e	c	e
clang 3.3	e	e	e	e	e	e	e	e	e	e	e	c	m	c	m	c	e	c	e
clang 3.4.2	e	e	e	e	e	e	e	e	e	e	e	c	m	c	m	c	e	c	e
clang 3.5.2	e	e	e	e	e	e	e	e	e	e	e	c	m	c	m	c	e	c	e
clang 3.6.2	e	e	e	e	e	e	e	e	e	e	e	c	m	c	m	c	e	c	e
clang trunk 36c8e6	e	e	e	e	e	e	e	e	e	e	e	c	m	c	m	c	e	c	e
icc (ICC) 14.0.1	e	e	e	e	e	e	e	m	e	e	m	e	m	c	m	e	e	c	e
Sun C++ 5.12	e	e	e	e	e	e	c	c	c	c	c	c	c	c	c	c	c	c	c
MSVC 8	e	e	e	e	e	e	e	e	e	e	e	c	c	c	c	c	e	c	e
MSVC 9	e	e	e	e	e	e	e	e	e	e	e	c	c	c	c	c	e	c	e
MSVC 10	e	e	e	e	e	e	e	e	e	e	e	c	c	c	c	c	e	c	e
MSVC 11	e	e	e	e	e	e	e	e	e	e	e	c	c	c	c	c	e	c	e
MSVC 12	e	e	e	e	e	e	e	e	e	e	e	c	m	c	m	c	e	c	e
MSVC 14	e	e	e	e	e	e	e	e	e	e	e	c	m	c	m	c	e	c	e
Digital Mars 8.57	e	e	e	e	e	e	e	e	e	e	e	e	c	c	c	e	e	c	e
Embarcadero 6.70 (x86)	e	e	e	e	e	e	c	c	c	c	c	c	c	c	c	c	c	c	c
Embarcadero 6.70 (x64)	e	e	e	e	e	e	e	e	e	e	e	c	m	c	m	c	e	c	e

Nomenclature: copy elided, object copied, object moved

Platforms:

- gcc/clang/icc Debian Squeeze/Jessie x64
- Sun C++ SunOs i386
- MSVC/Digital Mars/Embarcadero Windows 7 x64

Notes:

- All compilers had optimizations enabled.
- Embarcadero 6.70 (x64) is based on clang 3.1.
- For C++-11 capable compilers a move constructor was defined.
- The compilers decision for function inlining was not altered.

Function definitions

```
80  X urvo_single()
81  {
82    const bool param = true;
83    trace t("urvo_1");
84    return X();
85  }
```

←Jump to table

```
87  X urvo_two()
88  {
89    const bool param = true;
90    trace t("urvo_2");
91    if(param)
92      return X();
93    else
94      return X();
95  }
```

←Jump to table

```
97  X urvo_two_with_param(bool param)
98  {
99    trace t("urvo_two_with_param");
100   if(param)
101     return X();
102   else
103     return X();
104 }
```

←Jump to table

```
106 X urvo_with_exception_1(bool param)
107 {
108   trace t("urvo_with_exception_1");
109   if(!param)
110     throw std::exception();
111
112   return X();
113 }
```

←Jump to table

```
115 X urvo_with_exception_2(bool param)
116 {
117   trace t("urvo_with_exception_2");
118   if(param)
119     return X();
120   else
121     throw std::exception();
122 }
```

←Jump to table

```
124 X urvo_with_exception_3()
125 {
126   trace t("urvo_with_exception_3");
127   const bool param = true;
128   if(param)
129     return X();
130   else
131     throw std::exception();
132 }
```

←Jump to table

```
134 X nrvo_single_1()
135 {
136   trace t("nrvo_single_1");
137   X a;
138   return a;
139 }
```

←Jump to table

```
141 X nrvo_single_2()
142 {
143   trace t("nrvo_single_2");
144   {
145     X a;
146     return a;
147   }
148 }
```

←Jump to table

```
150 X nrvo_single_with_exception_1(bool param)
151 {
152   trace t("nrvo_single_with_exception_1");
153   X a;
154   if(!param)
155     throw std::exception();
156
157   return a;
158 }
```

←Jump to table

```
160 X nrvo_single_with_exception_2(bool param)
161 {
162   trace t("nrvo_single_with_exception_2");
163   X a;
164   if(param)
165     return a;
166   else
167     throw std::exception();
168
169    // Silence compilation error, does not count as an additional
170    // return statement as it is unreachable code
171   return a;
172 }
```

←Jump to table

```
174 X nrvo_single_with_exception_3()
175 {
176   trace t("nrvo_single_with_exception_3");
177   const bool param = true;
178   X a;
179   if(param)
180     return a;
181   else
182     throw std::exception();
183 }
```

←Jump to table

```
184 X nrvo_two_different_tern()
185 {
186   trace t("nrvo_two_different_tern");
187   const bool param = true;
188   X a, b;
189   return param ? a : b;
190 }
```

←Jump to table

```
192 X nrvo_two_different_if()
193 {
194   trace t("nrvo_two_different_if");
195   const bool param = true;
196   X a, b;
197   if(param)
198     return a;
199   else
200     return b;
201 }
```

←Jump to table

```
203 X nrvo_two_different_with_param_tern(bool param)
204 {
205   trace t("nrvo_two_different_with_param_tern");
206   X a, b;
207   return param ? a : b;
208 }
```

←Jump to table

```
210 X nrvo_two_different_with_param_if(bool param)
211 {
212   trace t("nrvo_two_different_with_param_if");
213   X a, b;
214   if(param)
215     return a;
216   else
217     return b;
218 }
```

←Jump to table

```
220 X nrvo_two_equal_tern()
221 {
222   trace t("nrvo_two_equal_tern");
223   const bool param = true;
224   X a;
225   return param ? a : a;
226 }
```

←Jump to table

```
228 X nrvo_two_equal_if()
229 {
230   trace t("nrvo_two_equal_if");
231   const bool param = true;
232   X a;
233   if(param)
234     return a;
235   else
236     return a;
237 }
```

←Jump to table

```
239 X nrvo_two_equal_with_param_tern(bool param)
240 {
241   trace t("nrvo_two_equal_with_param_tern");
242   X a;
243   return param ? a : a;
244 }
```

←Jump to table

```
246 X nrvo_two_equal_with_param_if(bool param)
247 {
248   trace t("nrvo_two_equal_with_param_if");
249   X a;
250   if(param)
251     return a;
252   else
253     return a;
254 }
```

←Jump to table

References

[wiki:rvo] http://en.wikipedia.org/wiki/Return_value_optimization

[rvo-cargo-cult] http://eatplayhate.me/2013/10/01/c-cargo-cults-rvo-and-copy-elision

[blog-knatten] http://blog.knatten.org/2011/08/26/dont-be-afraid-of-returning-by-value-know-the-return-value-optimization

[repository] https://github.com/t-b/cpp-copy-elision

[gcc-bug:51571] http://gcc.gnu.org/bugzilla/show_bug.cgi?id=51571